Haaga-Helia
ammattikorkeakoulu Oy

iOS-pohjaisen reittiopastinsovelluksen siirtäminen Android-käyttöjärjestelmälle

Jenni Wesslin

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Jenni Wesslin

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
iOS-pohjaisen reittiopastinsovelluksen siirtäminen Android-käyttöjärjestelmälle

Sivu- ja liitesivumäärä
24 + 3

Opinnäytetyön aiheena oli iOS-pohjaisen reittiopastinsovelluksen siirtäminen Android-käyttöjärjestelmälle, ja se toteutettiin toimeksiantona startup-yritykselle nimeltä Appsipaja oy. Tarve opinnäytetyölle syntyi toimeksiantajan tarpeesta toteuttaa, iOS-käyttöjärjestelmälle kehitetyn reittiopastinsovelluksen rinnalle, Android-käyttöjärjestelmälle sopiva vastaava sovellus. Tavoitteena oli kehittää alkuperäistä sovellusta vastaava sovellus Android-käyttöjärjestelmälle. Opinnäytetyön tuloksella tavoiteltiin laajemman käyttäjäkannan lisäksi lisäarvoa käyttäjille, joiden elämänlaatua pyritään sovelluksen avulla parantamaan. Projektin edetessä tarkoituksena oli myös kartuttaa tekijän yleistä ohjelmointiosaamista.

Opinnäytetyön teoriaosuudessa selvitetään syitä sille, miten kehitettävän sovelluksen ensimmäinen alusta valitaan. Valintaperusteiden lisäksi verrataan Android- ja iOS-alustojen eroja sovelluskehityksen näkökulmasta. Teoriaosuudessa kuvataan siirtoprosessin vaiheet, sekä siirtoprosessiin liittyvät hyödyt ja haasteet. Työn toteutusosa sisältää kuvauksen sovelluksen rakenteen suunnittelusta, sekä käyttöliittymänäkymien ja taustakoodissa käytettyjen menetelmien toteuttamisesta. Sovelluksen kehitystyöhön, dokumentoimiseen sekä opinnäytetyön kirjoittamiseen oli varattu aikaa noin 3,5 kuukautta keväällä 2018.

Opinnäytetyön pohdintaosiossa tarkastellaan työn lopputulosta ja pohditaan tehdyn työn jatkokehitysmahdollisuuksia. Tulosten lisäksi arvioidaan työn hyödyntämiskelpoisuutta sekä arvioidaan tekijän omaa oppimista ja osaamisen kehittymistä. Dokumentoinnissa käytetään runsaasti kuvakaappauksia kehitysprosessin havainnollistamiseksi.

Opinnäytetyön produktina syntyi Android-sovellus, joka on yhtä ominaisuutta lukuun ottamatta valmis alfajulkaisuvaiheeseen. Produktia voidaan tulevaisuudessa hyödyntää kehittämällä puuttuvat ominaisuudet valmiiksi.

Asiasanat
Android, iOS, mobiilisovellukset, ohjelmistokehitys, avoin lähdekoodi

Sisällys

1 Johdanto

Android-laitteet hallitsevat nykyisiä mobiililaitemarkkinoita lähes 90 prosentin osuudella. Menestystä tavoittelevan mobiilisovelluksen tulee olla saatavilla mahdollisimman laajalle käyttäjä- ja laitekannalle, etenkin Android- ja iOS-laitteille, sillä nuo kaksi mobiilikäyttöliittymää hallitsevat nykymarkkinoita kiihtyvällä tahdilla (Statista 2018). Sovelluksen on lisäksi vastattava entistä vaativampien käyttäjien tarpeisiin saumattomasti ja viiveettömästi.

Ellei sovelluksen ensimmäistä versiota ole kehitetty monialustaiseksi, eli usealla eri käyttöjärjestelmällä toimivaksi ratkaisuksi, on sovelluksen menestyksen kannalta tärkeää harkita sovelluksen saatavuuden laajentamista muillekin alustoille. Sovelluksen siirtäminen toiselle alustalle, tässä tapauksessa iOS-käyttöjärjestelmältä Android-käyttöjärjestelmälle, on ajankohtainen toimenpide sovelluksen elinkaaressa ja kehityksessä.

1.1 Tavoitteet, työn rakenne ja rajaus

Tämä opinnäytetyö toteutetaan toimeksiantona Appsipaja oy:lle. Tavoitteena on kehittää reittiopastinsovellus Android-käyttöjärjestelmälle, toimeksiantajan olemassa olevan iOS-sovelluksen rinnalle. Sovelluksen beetatestausvaihe on juuri käynnistynyt ja sovellus on kerännyt jo paljon kiinnostusta testaajien keskuudessa, joten Android-version kehittäminen on ajankohtainen ja looginen seuraava askel sovelluksen elinkaaressa. Työn tuloksella tavoitellaan laajemman käyttäjäkunnan lisäksi lisäarvoa käyttäjille, joiden elämänlaatua pyritään sovelluksen avulla parantamaan.

Opinnäytetyön toiminnallinen osuus koostuu sovelluksen kehittämisestä ja teoriaosuus sovelluksen kehitysvaiheiden kuvaamisesta. Teoriaosuudessa selvitetään syitä sille, miten kehitettävän sovelluksen ensimmäinen alusta valitaan. Valintaperusteiden lisäksi verrataan Android- ja iOS-käyttöliittymien eroja sovelluskehityksen näkökulmasta. Teoriaosuudessa kuvataan siirtoprosessin vaiheet, sekä siirtoprosessiin liittyvät hyödyt ja haasteet. Sovelluksen kehitysprosessista kertovassa osiossa kuvataan keskeisimpien käyttöliittymänäkymien sekä taustakoodissa käytettyjen menetelmien toteuttaminen. Lopuksi tarkastellaan työn lopputulosta, pohditaan tehdyn työn jatkokehitysmahdollisuuksia ja hyödyntämiskelpoisuutta sekä arvioidaan tekijän omaa oppimista ja osaamista.

Projektin tarkoituksena ei ole korvata olemassa olevaa iOS-sovellusta. Se ei myöskään tarjoa yksiselitteistä ratkaisua sovelluksen siirtämiseen toiselle alustalle, vaan keskittyy löytämään ongelmaan toimivan ratkaisun tekijän taitoja ja osaamista hyödyntämällä.

Alkuperäisen suunnitelman mukaan Android-sovellus oli tarkoitus toteuttaa alfatestausvaiheeseen ja Google Play -sovelluskaupassa julkaisuun saakka. Sovelluksen reittikartan toiminnallisuutta ei kuitenkaan ehditty rakentamaan täysin loppuun asti; reittikartan toiminnallisuus päätettiin rakentaa jäljellä olevan ajan puitteissa niin, että opastetekstit näkyvät tekstimuotoisena opastinpisteitä painamalla. Sovelluksen jatkokehittämisestä vastaa toimeksiantaja.

1.2 Keskeisimmät käsitteet

Käyttöjärjestelmä	Tietokoneen ohjelma, joka mahdollistaa muiden ohjelmien ja sovellusten käytön koneella. Käyttöjärjestelmä toimii raudan ja ohjelmien välissä, mahdollistaen sovellusten ja muiden ohjelmien yhteistyön koneen komponenttien kanssa.
Android OS	Open Handset Alliance -yhtymän (OHA) kehittämä Linux-pohjainen, avoimen lähdekoodin mobiilikäyttöjärjestelmä. Kehittämiseen osallistuivat Google-yhtiön lisäksi muun muassa HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia ja Wind River Systems.
iOS	Apple-yrityksen mobiilikäyttöjärjestelmä iPhone-, iPad- ja muille Apple-laitteille. Esiintyi aiemmin nimellä iPhone OS, nimi muutettiin vuonna 2010 kuvaamaan paremmin käyttöjärjestelmän kehittyvää ja kasvavaa laitekantaa.
Beetatestaus	Sovelluksen toimintaa voidaan testata valikoidulla testiryhmällä, kun sovellus on saatu kehitettyä siihen pisteeseen, että ominaisuuksien testaaminen on ajankohtaista. Testausvaiheen tarkoituksena on löytää viimeiset mahdolliset virheet sovelluksesta, ennen sovelluksen virallista julkaisua. Beetatestausta edeltää yleensä alfatestivaihe, jolloin pieni joukko alan asiantuntijoita testaa sovelluksen toimintaa suurimpien virheiden varalta. Beetatestauksessa testaajat koostuvat pääasiassa tavallisista sovelluksen käyttäjistä.
Google Play -kauppa	Google-yhtiön omistama digitaalinen sisältöpalvelu, jonka kautta käyttäjä voi ladata ilmaisia ja maksullisia sovelluksia Android-mobiililaitteelle.

App Store	Apple-sovelluskauppa, jonka kautta käyttäjä voi ladata ilmaisia ja maksullisia sovelluksia iOS-mobiililaitteelle.
UI	Käyttöliittymä (user interface). Laitteen tai ohjelmiston osa, jonka kautta käyttäjä käyttää laitetta tai ohjelmistoa.
API	Ohjelmointirajapinta (application programming interface). Ohjelmointirajapinta toimii ohjelmiston komponenttien välillä kommunikaatioalustana, jonka avulla ohjelmat pystyvät tekemään pyyntöjä ja vaihtamaan tietoa keskenään.
Resoluutio	Kuvantoistolaitteen tarkkuus eli erottelukyky.
Firebase	Mobiilisovelluksille tarkoitettu tietokanta, jota voidaan hyödyntää myös web-sovelluksissa tietojen käsittelyssä ja todentamisessa.
GitHub	Verkkosivusto, jota Git-versionhallintaa käyttävät ohjelmistoprojektit voivat käyttää muun muassa ohjelmistokoodien tallennuspaikkana.
Mapbox	Vapaan lähdekoodin sijaintitieto alusta mobiili- ja web-sovelluksille.

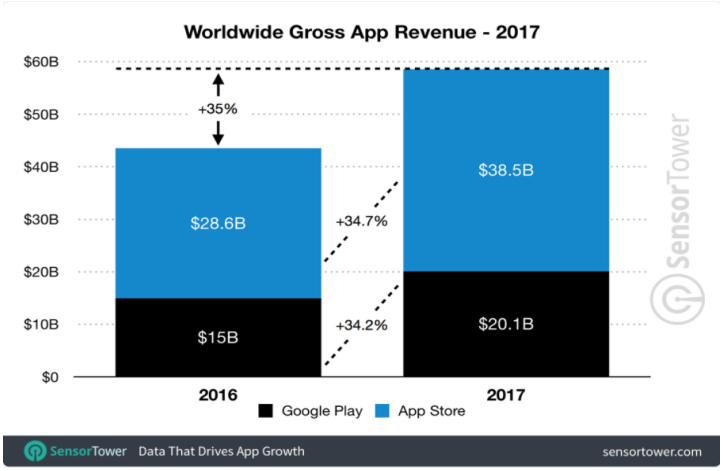
2 IOS-sovelluksesta Android-sovellukseksi

Idea hyödyllisestä sovelluksesta syntyy yleensä tarpeesta helpottaa ihmisten arkielämää tai parantaa sitä jollain tavalla. Kun yritys tai yksityishenkilö on tehnyt päätöksen sovelluksen kehittämisestä, on aika päättää, mille alustalle sovelluksen ensimmäinen versio tulisi kehittää.

Bessarabovan (12.5.2017a) mukaan monet suositut sovellukset, kuten esimerkiksi Instagram, Worms 3 ja Clash of Clans, ovat aloittaneet taipaleensa iOS-pohjaisina sovelluksina. Yhtenä syynä tähän ilmiöön Bessarabova (12.5.2017a) pitää iPhone-käyttäjien avointa ostoskäyttäytymistä Apple-yhtiön App Store -sovelluskaupassa. Hänen mukaansa Apple-käyttäjät ovat halukkaampia maksamaan käyttämistään sovelluksista, joten he tuottavat sovelluksille Android-käyttäjiä helpommin myyntituloa.

2.1 Sovelluksen ensialustan valintaan vaikuttavat tekijät

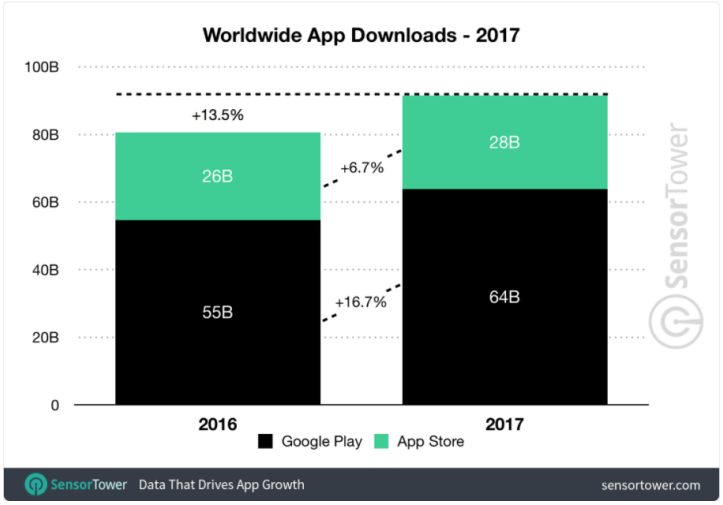
App Store -sovelluskaupan maailmanlaajuinen liikevaihto oli vuonna 2017 sovelluksista saatujen tulojen osalta 38,5 miljardia euroa. Google Play -sovelluskaupan tuotot jäivät reilusti alle App Store -sovelluskaupan tuloksen, 20,1 miljardiin euroon. (Nelson 5.1.2018.) Kaavio maailmanlaajuisen sovelluskaupan bruttotuloista (kuvio 1) antaa selkeän kuvan App Store -sovelluskaupan ylivoimaisesta johdosta sovelluksista saatujen tuottojen osalta.



Kuvio 1. Maailmanlaajuisen sovelluskaupan bruttotulot vuonna 2017 (Nelson 5.1.2018)

Apple-laitteiden omistajat vaikuttaisivat käyttävän enemmän rahaa sovelluskaupassa kuin Android-laitteiden käyttäjät. Sovellusten latausmäärissä Google Play -sovelluskauppa hallitsi tilastoja 27,2 miljardilla latauksella, kun App Store -sovelluskaupan tulos jäi 8,3 miljardiin lataukseen (kuvio 2). Näitä kaavioita vertailtaessa käy ilmi, että Apple-laitteiden käyttäjät lataavat sovelluksia verraten vähemmän, mutta maksavat käyttämistään sovel-

luksista, kun taas Android-laitteiden käyttäjät suosivat ilmaisia sovelluksia ja lataavat niitä laitteilleen enemmän. Taloudellisen menestyksen tavoittelun kannalta kannattavampi ratkaisu on siis kehittää sovellus ensin iOS-järjestelmälle.



Kuvio 2. Maailmanlaajuiset sovelluslatausmäärät vuonna 2017 (Nelson 5.1.2018)

Bessarabovan (12.5.2017a) mukaan toinen merkittävä syy iOS-version kehittämiseen ennen Android-versiota on Android-laitemarkkinoiden sirpaloituminen. Laitemarkkinoiden sirpaloitumisella tarkoitetaan sitä, että saatavilla olevien Android-laitteiden ja käyttöjärjestelmäversioiden kirjo on hyvin laaja. Android-sovellusten kehittäminen vaatii huomattavasti enemmän aikaa testauksen ja varsinaisen kehitystyön osalta, sillä sovelluksen tulee toimia mahdollisimman monenlaisilla laite- ja käyttöjärjestelmä muunnelmilla. Apple-laitteilla tätä ongelmaa ei esiinny, sillä tuettuja iOS-käyttöjärjestelmäversioita on Android-versioihin nähden vähän. (Bondarenko 2017.)

Tuettujen käyttöjärjestelmäversioiden pienehkö sirpaloituminen helpottaa ja nopeuttaa sovelluksen kehitystyötä huomattavasti, jolloin iOS-version kehittäminen ennen Android-versiota on hyvin perusteltu ja taloudellisesti järkevä ratkaisu, kun halutaan minimoida taloudelliset investoinnit ja kehitystyöhön käytettävä aika. Bondarenkon (2017) mukaan helpoin ratkaisu olisi luoda natiivisovelluksien sijaan monialustainen toteutus, jolloin saataisiin kehitettyä yhdellä koodipohjalla monelle käyttöjärjestelmälle sopiva sovellus. Hän käyttää esimerkkinä React Native -kehystä, jonka kehittivät Facebook-sovelluksen kehittäjät. Se mahdollistaisi koodin toteuttamisen valtaosin JavaScript -ohjelmointikielellä, ja laitekohtaisten osioiden kirjoittamisen Java- tai Swift-ohjelmointikielellä. Bondarenko ehdottaa, että tällä tavoin koodista pystyttäisiin uudelleenkäyttämään noin 50-80 prosenttia, sovelluksesta riippuen. Tämän opinnäytetyön puitteissa natiivisovelluksen kehittäminen on monialustaisen sovelluksen sijaan parempi ratkaisu, sillä tekijällä on kokemusta ainoastaan Android-ohjelmoinnista, eikä uudelle iOS-sovellukselle ole tällä hetkellä tarvetta.

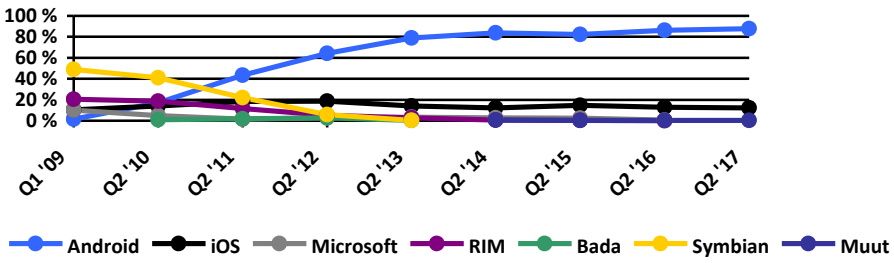
Sovelluksen siirtämisprosessissa on otettava huomioon muutamia ulkonäöllisiä sekä toiminnallisia seikkoja, jotta uusi sovellus vastaisi ominaisuuksiltaan ja ulkonäöltään alkuperäistä sovellusta, olematta kuitenkaan kopio alkuperäisestä sovelluksesta. Alkuperäisen sovelluksen kopioiminen toiselle alustalle ei ole mahdollista, kannattavuudesta puhumattakaan, sillä alustoina Android ja iOS ovat hyvin erilaiset. Molemmat alustat vaativat tietynlaisten suuntaviivojen noudattamista sovelluskehityksessä. (Thinkmobiles 2017; Bondarenko 2017.)

Sovelluksen siirtäminen yhdeltä alustalta toiselle pienentää uuden sovelluksen kehittämiskuluja, laajentaa sovelluksen potentiaalista käyttäjäkantaa ja luo taloudellista lisähyötyä sovelluksen omistajalle. Sovelluksen siirtäminen tuottaa hyödyllisen ja tuottavan lopputuloksen, kun alustakohtaiset vaatimukset ja ominaisuudet otetaan huomioon uuden sovelluksen kehitysprosessia suunniteltaessa. (Bessarabova 12.5.2017a.)

2.2 Sovelluksen siirtämisprosessin hyödyt

Sovelluksen siirtämisellä tarkoitetaan tässä opinnäytetyössä prosessia, jossa olemassa oleva sovellus toteutetaan toiselle käyttöjärjestelmälle. Tarve sovelluksen siirtämiselle ilmenee yleensä silloin, kun sovellus on kehitetty lähelle julkaisuvaihetta yhdelle käyttöjärjestelmälle, ja menestystä halutaan tavoitella laajentamalla sovelluksen saatavuutta muillekin käyttöjärjestelmille.

Android ja iOS hallitsevat nykyisin mobiililaitemarkkinoita, joista Android-laitteiden osuus on noin 88 prosenttia ja iOS-laitteiden osuus noin 12 prosenttia (kuvio 3). Android-käyttöjärjestelmälle laajentaminen on looginen siirtymä sovellukselle, jolle tavoitellaan mahdollisimman laajaa käyttäjäkantaa ja mahdollisia taloudellisia lisähyötyjä. Android-version puuttuminen rajoittaa sovelluksen potentiaalista käyttäjäkuntaa ja taloudellista tulosta merkittävästi.



Kuvio 3. Android- ja iOS-laitteiden markkinaosuuksien kehitys vuodesta 2009 vuoteen 2017 (Statista 2018)

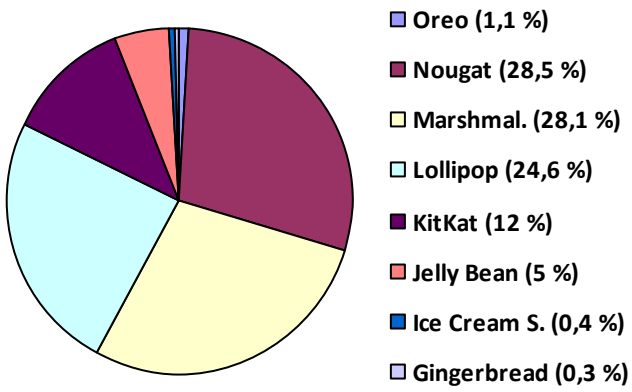
Sovelluksen siirtäminen on ajallisesti ja rahallisesti verraten edullisempi ratkaisu kuin sovelluksen kehittäminen alusta alkaen uudestaan, sillä käyttöliittymän rakenne ja taustakoodin logiikka on jo mietitty valmiiksi alkuperäisessä sovelluksessa. Sovelluksen siirtäminen uudelle alustalle sisältää myös tiettyjä haasteita, jotka on otettava huomioon siirtämisprosessia suunniteltaessa. Huomioon otettavia asioita ovat esimerkiksi alustojen arkkitehtuuriset eroavaisuudet, ulkoasuun ja käytettävyyteen liittyvät yksityiskohdat, koodi ja sen komponentit, kohderyhmä ja bisnesmalli, testaus sekä sovelluksen julkaisu Google Play -kaupassa. Sovelluksen siirtäminen mahdollistaa käyttäjäkokemuksen rikastamisen alustakohtaisilla ominaisuuksilla ja sovelluksen räätälöimisen omanlaiseksi kokonaisuudeksi, jonka ei tarvitse orjallisesti noudattaa alkuperäisen sovelluksen toimintaa ja ulkoasua. (Bessarabova 12.5.2017a; ThinkMobiles 2017.)

2.3 Sovelluksen siirtämisprosessin haasteet

Ohjelmointikäytännöt eroavat alustojen välillä, sillä Android sekä iOS hyödyntävät eri ohjelmointikieliä sovelluskehityksessä. Android-sovellukset kehitetään pääosin Java- tai Kotlin-ohjelmointikielellä, iOS-sovellukset puolestaan Swift-ohjelmointikielellä. Alkuperäisessä sovelluksessa käytetyt ratkaisut eivät ole tästä johtuen suoraan sovellettavissa toiselle alustalle, eikä valmiita ratkaisuja voi kopioida suoraan alustalta toiselle. (Bondarenko 2017.)

Sovelluksen ohjelmoiminen Java-ohjelmointikielellä vaatii suuren koodimäärän kirjoittamista, kun taas iOS:n käyttämä Swift-ohjelmointikieli tarjoaa nopeamman ja tehokkaamman tavan kirjoittaa koodia. Swift on Java-kieltä uudempi ohjelmointikieli, joten sen uudemmat ja kehittyneemmät ohjelmointimenetelmät mahdollistavat tehokkaamman sovelluskehityksen Java-kieleen verrattuna. Tilanne saattaa kuitenkin muuttua lähitulevaisuudessa: Android Studio 3.0 -version myötä Java-kielen rinnalle on tullut uusi, täysin tuettu ohjelmointikieli, Kotlin. Kotlin-kielen tehokkuus sekä kehittyneempi suorituskyky vastaavat Swift-kielen ominaisuuksia, joten se saattaa syrjäyttää Java-kielen ensisijaisena Android-työkaluna lähitulevaisuudessa. (Bessarabova 21.2.2017b; Verma 15.10.2017.)

Tuettujen Android-käyttöjärjestelmäversioiden suuri määrä aiheuttaa lisää haasteita kehitysprosessille; tällä hetkellä yli 10 prosentin käyttäjäjakaumalla olevia käyttöjärjestelmäversioita on viisi: KitKat, Lollipop, Marshmallow, Nougat ja Oreo (kuvio 4).



Kuvio 4. Android-käyttöjärjestelmäversioiden käyttöjakauma (Android Developers 2018a)

Näin suuri jakauma monen eri alustan kesken tarkoittaa käytännössä sitä, että sovelluksen tulisi toimia ainakin neljällä suosituimmalla käyttöjärjestelmällä moitteettomasti, jotta sovellus tavoittaisi mahdollisimman paljon käyttäjiä. Vanhempien käyttöjärjestelmien tukeminen rajoittaa uusimpien ominaisuuksien käyttämistä, joten siirtämisprosessissa on tehtävä päätöksiä sen suhteen, mitkä ominaisuudet ovat tärkeitä sovelluksen toiminnan kannalta.

2.4 Käyttöjärjestelmien väliset ulkoasuerot

Android sekä iOS omaavat omanlaisensa suuntaviivat käyttöliittymiensä suhteen, joita on noudatettava ulkoasua suunniteltaessa. Käyttöliittymän rakennetta ei voi siirtää sellaisenaan alustalta toiselle, sillä Android- ja iOS-käyttöjärjestelmien sovellusarkkitehtuurit eroavat rakenteellisesti toisistaan merkittävästi. Yksi suurimmista eroista ilmenee elementtien sijainneissa. iOS-sovellukset ovat rakenteeltaan yksinkertaisia, eli jokainen käyttöliittymäelementti toimii kaksiulotteisessa tilassa ja näkymät ovat melko yksinkertaisia ja ovat toisiinsa nähden samalla tasolla; Android-sovelluksien käyttöliittymäelementit puolestaan noudattavat moniulotteista asettelua. Listaelementeissä Android käyttää mukautettuja ikoneja eikä käytä nuolia, kuten iOS:n listaelementit. Android-sovellusten navigointipalkki on korkeampi ja otsikko on vasemmalla, iOS-sovellusten navigointipalkki on ohuempi ja otsikko keskellä palkkia. Valitsimet ja fontit eroavat myös toisistaan. Lisäksi ikonit ovat Android- ja iOS-sovelluksissa erilaiset; iOS:n ikonien on noudatettava tiukempia ulkonäkövaatimuksia, kun Android-sovellusten ikonit ovat muokattavissa monella eri tavalla. (Bessarabova 12.5.2017a; Bondarenko 2017.)

iOS-laitteilla on olemassa yksi fyysinen painike laitteen alaosassa, joka palauttaa käyttäjän kotinäkymään (home screen). Android tarjoaa fyysiseen navigointiin kolme pääpainiketta: koti-painikkeen, takaisin-painikkeen sekä moniajo-painikkeen (multitasking), jonka

avulla käyttäjä voi hallinnoida useita sovelluksia samaan aikaan. Fyysisten painikkeiden sijainnit vaikuttavat myös käyttöliittymän suunnitteluun. iOS-sovellukset koostuvat usein vaaka- sekä pystytasoisista elementeistä, kun Android-sovellukset suosivat enimmäkseen vaakatasoisia elementtejä. Lisäksi on otettava huomioon navigointipalkkien sijainti; Android-laitteiden fyysiset navigointipainikkeet sijaitsevat laitteen alalaidassa, joten iOS-sovelluksille ominaista alalaidan navigointia ei kannata toteuttaa Android-sovelluksissa, sillä käyttökokemus saattaa kärsiä liian lähelle toisiaan sijoitetuista navigointipainikkeista. (Bondarenko 2017.)

Siirtämisprosessissa tulee myös huomioida Android-laitteiden kokoerojen moninaisuus ja niiden vaikutus sovelluksen näkymiseen erikokoisilla näytöillä. Apple-laitteilla tämä ei ole ongelma, sillä laite-eroja on Android-laitteisiin verraten vähän. IPhone-laitteiden tyypillisimmät näyttökoot vaihtelevat 4.0 - 5.5 tuuman välillä, kun Android-laitteilla näyttöjen kokojen vaihtelevuus on hyvin suurta. 5-7 yleisimmän resoluutiomuunnelman huomioon ottaminen varmistaa sovelluksen toiminnan samankaltaisilla mutta erikokoisilla näytöillä. (Bondarenko 2017.)

2.5 Sovelluksen siirtämisprosessin vaiheet

Siirtämisprosessissa on aluksi analysoitava iOS-sovelluksen toiminta, sen ominaisuudet sekä ulkoasun ominaispiirteet. On myös määritettävä mahdolliset ongelmakohdat ja mietittävä niiden ratkaiseminen tai optimointi Android-versiossa. Taustakoodin erityispiirteiden, käytettyjen työkalujen ja ohjelmistokehysten (framework) yhteensopivuus tulee varmistaa, jotta sovelluksen siirtäminen onnistuisi odotetulla tavalla. Yhteensopivuustekniset ongelmat liittyen koodiin, kirjastoihin ja kolmannen osapuolen palveluihin on arvioitava ja valittava niiden perusteella sopiva tekninen yhdistelmä, jotta sovelluksen siirtäminen olisi tehokasta. (ThinkMobiles 2017.)

Laaduntarkkailu kehitysprosessin aikana on tärkeää laadun ja virheettömyyden takaamiseksi. Sovelluksen jatkuva testaaminen koko kehitysprosessin ajan esimerkiksi emulaattorilla eli virtuaalisella laitteella tai oikealla laitteella on tärkeää. Ennen varsinaista julkaisua Google Play -kaupassa on suoritettava ainakin alfa- ja beetatestausvaiheet, jolloin sovelluksen toimintaa testataan ensin suurimpien virheiden löytämiseksi (alfatestaus) ja myöhemmin yleisen toiminnallisuuden kannalta (beetatestaus). Kun sovellusta on testattu tarpeeksi, voidaan se julkaista Google Play -kaupassa ja aloittaa sovelluksen markkinointikampanjat. (ThinkMobiles 2017.)

3 Sovelluksen toiminnan kuvaus ja toteutusprosessi

Route Pepper on reittiopastinsovellus, jonka tarkoituksena on helpottaa esimerkiksi lenkkeilijöiden, pyöräilijöiden ja turistien elämää tarjoamalla helppokäyttöinen ratkaisu käyttäjien omien reittien suunnitteluun ja kulkemiseen. Reitit suunnitellaan ja toteutetaan web-sovelluksessa, jonne käyttäjä kirjautuu omilla tunnuksillaan. Reitille piirretään reittiviivat ja reitille asetetaan opastinpisteitä eli Pep-pisteitä, joihin kirjoitetaan opastetekstejä reitin kulkemisen helpottamiseksi. Web-sovelluksessa luodut reitit tallennetaan Firebase-tietokantaan; tallennetut reitit löytyvät mobiilisovelluksesta käyttäjän kirjautuessa sisään omilla tunnuksillaan. Käyttäjä voi kulkea valitsemaansa reittiä ja kuulla opastinpisteille asetetut opastetekstit puheena kuulokkeidensa kautta; laitetta ei tarvitse katsoa kertaakaan suorituksen aikana ja käyttäjä voi keskittyä rauhassa suoritukseensa.

Android-sovelluksen kehityksessä lähdettiin liikkeelle siitä periaatteesta, että sovelluksen toiminnallisuuden tulisi vastata alkuperäistä toteutusta, mutta ulkoasu ja käyttöliittymä tulisi suunnitella ja toteuttaa Android-ympäristöön sopiviksi. Ennen varsinaisen kehitysprosessin aloittamista oli tarpeellista suorittaa muutamia alkutoimenpiteitä varsinaisen kehitysprosessin helpottamiseksi.

Alkutoimenpiteinä suoritettiin

- sovelluksen rautalankamallin rakentaminen
- lähdekoodiin tutustuminen pintapuolisesti
- Swift-ohjelmointikielen tutoriaaleihin tutustuminen
- uudelleenkirjoittamiseen liittyvän aineiston kokoaminen ja tutkiminen
- projektinhallintatyökalun valmistelu
- Android Studio -projektin luominen
- projektin yhdistäminen toimeksiantajan luomaan GitHub-ohjelmavarastoon.

Sovelluksen ulkoasusta ja toiminnasta luotiin rautalankamalli Justinmind-käyttöliittymätyökalun avulla. Rautalankamallin tavoitteena oli hahmotella sovelluksen mahdollista ulkoasua ja toimintaa. Rautalankamalli vastasi ulkoasultaan iOS-sovelluksen ulkoasua, mutta kehitettävän sovelluksen ulkoasussa päätettiin noudattaa uutta, kolmannen osapuolen tarjoamaa ulkoasusuunnitelmaa. Seuraavaksi oli luotava profiili Route Pepper -web-sovelluksella ja luotava yksi oma reitti sen avulla, jotta saatiin paikkansapitävää testimateriaalia sovelluksen kehittämistä ja toiminnallisuuksien testaamista varten. Alkuperäisen sovelluksen toimintaan tutustuttiin kokeilemalla sovellusta toimeksiantajan lainaamalla iPhone-laitteella.

Lähdekoodiin tutustuttiin ennen projektin aloittamista pintapuolisesti; ensimmäisessä ohjauskokouksessa perusteellinen lähdekoodiin tutustuminen päätettiin siirtää osaksi tausta-

koodin kehittämistä, jotta lähdekoodia voitaisiin hyödyntää tehokkaammin silloin kun sitä tarvitaan. Swift-tutoriaaleihin perehdyttiin sen verran, että lähdekoodin logiikka ja toiminnallisuus tulisi tutummaksi. Sovelluksen siirtämistä käsittelevä aineisto koottiin ja siihen tutustuttiin huolellisesti. Ennen varsinaisen kehitysprosessin aloittamista oli otettava selvää, mitä sovelluksen siirtämisprosessissa tulee ottaa huomioon. Tällä tavoin mahdolliset ongelmat otettiin huomioon jo ennen sovelluksen kehittämistä.

Android Studio -ohjelmointiympäristöön luotiin projekti, jonka alhaisimmaksi tuetuksi Android-versioksi (minSdkVersion) valittiin API-taso 21, jotta kaikki sovelluksen ominaisuudet toimisivat varmasti ongelmitta. Valmis projektipohja yhdistettiin toimeksiantajan GitHub-ohjelmavarastoon (repository). Lopuksi projektia varten luotiin projektinhallintatyökalu Trello-projektinhallintajärjestelmää apuna käyttäen. Seuraavissa alaluvuissa kuvataan kehittämisprosessin toteutusvaiheet, tulokset ja kehitystyön tuloksena syntynyt sovellus.

3.1 Ulkoasun toteutus

Kehitysprosessi aloitettiin sovelluksen ulkoasun eli näkymien ulkoasujen (activity layout) luomisella. Ensin oli määritettävä sovelluksen käynnistysnäkymän ulkoasu (launcher activity layout). Käynnistysnäkymällä voidaan antaa sovellukselle aikaa valmistella tarvittavat komponentit, ja samalla mainostaa sovellusta käyttämällä näkymän ulkoasun kuvana vaikkapa sovelluksen logoa.

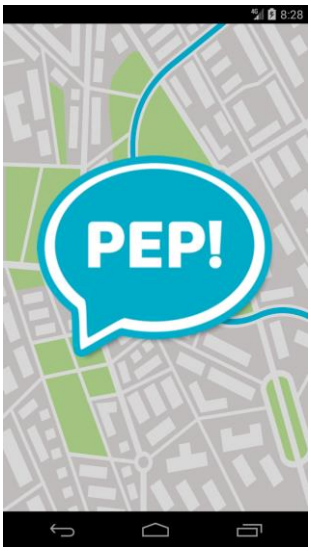
Usein sovelluksen käynnistyminen kestää pienen hetken, kun sovellus käynnistetään ensimmäisen kerran (cold start). Kylmäkäynnistyksen aikana järjestelmällä on kolme tehtävää hoidettavana; sen täytyy ladata ja käynnistää sovellus, näyttää tyhjä aloitusikkuna heti käynnistämisen jälkeen, sekä luoda sovellusprosessi. Sovellusprosessi hoitaa päänäkymän näyttämisen ja näkymän komponenttien täyttämisen. Tämäntyyppinen käynnistys on suuri haaste käynnistysajan minimoinnin kannalta, koska järjestelmällä ja sovelluksella on enemmän työtä kuin muissa käynnistystiloissa. (Android Developers 2018b.)

3.1.1 Alkunäkymä ja GPS-käyttöoikeus -näkymä

Sovelluksen käynnistysnäkymäksi luotiin alkunäkymä (splash screen) valkoisen alkuruudun välttämiseksi sekä ammattimaisemman ilmeen antamiseksi (kuva 1). Alkunäkymä luotiin ilman ulkoasutiedostoa (layout resource file); näkymälle ilmoitetaan ulkoasun teema AndroidManifest -tiedostossa. Alkunäkymä saa teeman tyylitiedoston kautta, missä teeman taustakuvaksi on määritelty piirtotiedosto (drawable resource file), joka puolestaan

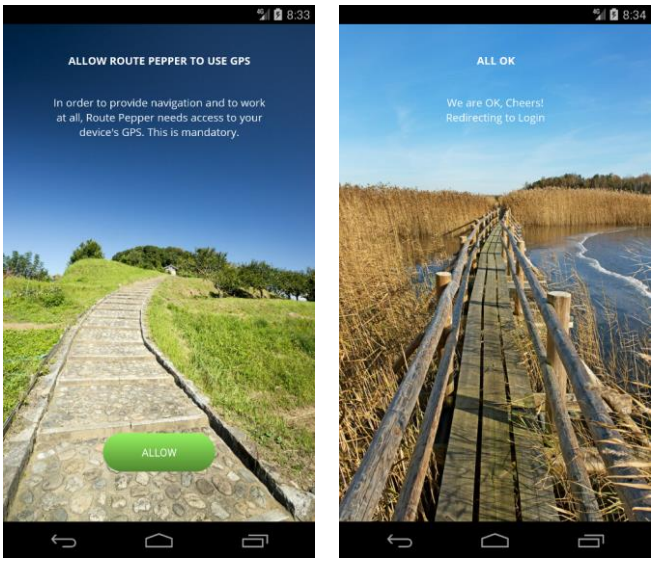
sisältää Pep-kupla -kuvan ja karttataustan. Näkymän ainoa tarkoitus on ohjata käyttäjä päänäkymään heti, kun sovellus on latautunut kokonaan.

Sinhalin (14.01.2017) mukaan tämä on tehokas ja nopea tapa tehdä alkunäkymä, kun sen halutaan näkyvän vain silloin, kun sovelluksen taustaprosessit ovat kesken. Jos sovellus on jo puhelimen välimuistissa, alkunäkymä häviää melkein saman tien.



Kuva 1. Alkunäkymä

Alkunäkymän jälkeen rakennettiin GPS-käyttöoikeus -näkymä (GPS-permission screen) (kuva 2). Näkymän avulla pyydetään käyttäjää kytkemään puhelimen GPS-paikannus päälle. GPS-paikannus on sovelluksen toiminnan kannalta elintärkeä ominaisuus, sillä sovelluksen toiminta perustuu käyttäjän sijainnin paikantamiseen; kartta ja reitit eivät toimi oikein, jos paikannusominaisuutta ei ole kytketty päälle.



Kuva 2. GPS-käyttöoikeus -näkymä

Näkymälle on asetettu oma taustakuva, seliteteksti ja painike ominaisuuden hyväksymistä varten. Painike saa tyylinsä piirtotiedostosta, missä sen taustaksi on määritelty vihreä liukuväri ja pyöristetyt kulmat. Selitetekstille ja otsikolle haetaan fontit tarvikekansion (assets) fonttitiedostosta. Kun käyttäjä on antanut sovellukselle luvan GPS-paikannukseen, käyttäjä siirretään rekisteröinti- ja sisäänkirjautumisnäkymään.

3.1.2 Valintanäkymä

Seuraavaksi rakennettiin valintanäkymä rekisteröintiin ja sisäänkirjautumiseen. Tämä näkymä antaa käyttäjälle mahdollisuuden joko kirjautua sisään olemassa olevilla tunnuksilla, tai rekisteröityä sovelluksen käyttäjäksi osoitteessa www.routepepper.com (liite 1a). Luo tili -painike (create account) avaa Route Pepper -rekisteröitymislomakkeen selaimessa; uusi käyttäjä voi luoda itselleen tilin ja kirjautua sen jälkeen tunnuksilla sisään. Kirjaudu sisään -painike (log in) avaa sisäänkirjautumisnäkymän.

Näkymän taustakuva on asetettu tyylitiedostossa perusteemaan (BasicTheme), jonka kautta suurin osa sovelluksen näkymistä saa taustakuvansa. Näkymälle on määritelty teema AndroidManifest- tiedostossa. Kuplalogo on asetettu näkymän ulkoasutiedostossa kuvanäkymä -komponenttiin (ImageView).

Painikkeet ovat kustomoituja peruspainikkeita, joille on määritetty tyylit piirtotiedostossa; niiden taustaksi on määritelty liukuväri ja pyöristetyt kulmat. Painikkeiden kuvaikonit on asetettu painikkeiden vasempaan reunaan. Kuvaikonit tulevat piirtotiedostosta.

3.1.3 Sisäänkirjautumisnäkymä

Tässä näkymässä käyttäjää pyydetään syöttämään www.routepepper.com-osoitteessa luomansa tilin sähköpostiosoite ja salasana (liite 1b). Salasanakentän oikeassa reunassa oleva silmäikoni-painike näyttää ja piilottaa kirjoitetun salasanan painiketta painettaessa. Muista minut -valintaruutu salasanakentän alla antaa käyttäjälle mahdollisuuden tallentaa laitteen muistiin tekstikenttiin syötetyt arvot, kun valintaruutu on aktivoitu. Sisäänkirjautumispainiketta painettaessa sovellus ottaa yhteyden tietokantaan ja hakee syötettyjä tietoja vastaavan tilin, varmentaa käyttäjän, kirjaa käyttäjän sisään ja siirtää käyttäjän lopuksi reittilistanäkymään.

Tämä näkymä luotiin Android Studio -ohjelmointiympäristön valmiilla kirjautumistoimintomallilla (Login Activity), jota muokattiin sopimaan paremmin sovelluksen toiminnallisiin vaatimuksiin. Taustan, logon ja sisäänkirjautumispainikeen tyylit määritetään tässä näkymässä samalla tavalla kuin valintanäkymässä, sähköposti- ja salasanakenttien sekä Muis-

ta minut -valintaruudun tyylit tulevat näkymän ulkoasutiedostosta. Salasanakentän silmäikoni sekä tekstikenttien kuvaikonit on asetettu komponentteihin samalla tavalla kuin valintanäkymän painikkeiden ikonit.

3.1.4 Reittilistanäkymä

Sisäänkirjautumisnäkymän valmistuttua rakennettiin reittilistanäkymä, jonka tarkoituksena on näyttää kaikki käyttäjän luomat reitit listaelementteinä ja antaa käyttäjälle mahdollisuus valita haluamansa reitti listasta (liite 2a). Reiteistä näytetään nimi, luontipäivä ja pituus metreinä. Listaelementit toimivat painikkeiden tavoin; haluttua reittiä painamalla käyttäjä siirtyy karttanäkymään. Käyttäjä voi myös avata navigointivalikon joko yläpalkin hampurilaisikonista tai näytön vasemmasta reunasta liu'uttamalla. Navigointivalikko sulkeutuu painamalla valikon ulkopuolelta tai liu'uttamalla valikkoa takaisin vasempaan reunaan. Navigointivalikosta on otettu pois käytöstä ne valikkokohdat, joissa ei vielä ole sisältöä. Asetukset-painike (Settings) avaa asetusnäkymän, Ohjeet-painike (Instructions) avaa sovelluksen pikaohjeet. Kirjaudu ulos-painike (Log out) kirjaa käyttäjän ulos ja siirtää käyttäjän takaisin sisäänkirjautumisnäkymään.

Tämä näkymä on rakennettu Android Studio -ohjelmointiympäristön tarjoaman navigointivalikko-mallin mukaan (Navigation Drawer Activity). Näkymän ulkoasutiedosto hakee näkymän komponentit eri ulkoasutiedostoista. Näkymän reittilistalla ja sen yläpalkilla, navigointivalikon sisällöllä ja sen otsakkeella on kaikilla omat ulkoasutiedostonsa, jotka yhdistetään näkymän omassa ulkoasutiedostossa. Reittilistan listaelementit saavat myös tyylin erillisestä ulkoasutiedostosta, jossa jokaiselle listaelementille asetetaan reitti-ikoni ja värillinen tekstikenttä.

Navigointivalikon otsakkeella on tausta, jonka se saa piirtotiedoston kautta. Otsake sisältää kuvanäkymä-komponentin jonka taustakuvana on sovelluksen logo, tekstikentän joka näyttää sovelluksen nimen, sekä toisen tekstikentän johon sovellus hakee käyttäjän sähköpostiosoitteen tietokannasta. Valikkokohdat tulevat valikko-resurssitiedoston (menu resource file) ulkoasutiedostosta, jossa jokaiselle valikkoelementille on määritelty otsikko ja ikoni.

3.1.5 Asetusnäkymä

Tämän näkymän toiminnallisuuden kehittäminen päätettiin jättää opinnäytetyöstä pois, sillä tämä näkymä ei ollut sovelluksen toiminnan kannalta tässä vaiheessa tärkeä. Näkymä rakennettiin, jotta käyttäjä pystyisi muokkaamaan sovelluksen ominaisuuksia ja kytkemään niitä päälle ja pois päältä (liite 2b). Ääniopasteiden ollessa pois päältä, sen hieno-

säätövalitsimet ovat piilossa. Kytkimen ollessa aktiivinen, valitsimet ovat näkyvissä. Yläpalkin nuolipainikkeesta käyttäjä pääsee takaisin reittilistaan.

Näkymän tausta ja yläpalkki noudattavat muiden näkymien tyyliä. Komponentit koostuvat kytkimistä ja hienosäätövalitsimista, joilla on taustat ja otsikot. Hienosäätövalitsimet ovat riippuvaisia Puheen etäisyys -kytkimestä (speech distance); hienosäätövalitsimet ovat piilotettuina silloin, kun kytkin ei ole aktiivisessa tilassa.

3.1.6 Ohjenäkymä

Tämä näkymä sisältää pikaohjeet sovelluksen käyttöön ja toimintaan (liite 3a). Viimeisellä ohjesivulla on painike täysversion ostamiseen, mutta sillä ei ole tällä hetkellä mitään toiminnallisuutta. Sovellus on toistaiseksi täysin ilmainen, toimeksiantajan on tarkoitus ottaa maksulliset ominaisuudet käyttöön tulevaisuudessa.

Näkymän sisällöt eli sivut vaihtuvat yhden ulkoasutiedoston sisällä näkymien selaamiseen tarkoitetussa komponentissa (ViewPager). Sivuja voi selata liu'uttamalla näkymiä vasemmalle tai oikealle. Komponentti on toteutettu Comptonin (31.7.2015) tarjoaman mallikoodin mukaan. Näkymän ylälaidassa olevasta ruksista käyttäjä voi sulkea ohjesivut ja palata takaisin reittilistanäkymään. Näkymän alalaidassa on sivuindikaattori (TabLayout), joka näyttää nykyisen sivun sekä sivujen kokonaismäärän. Sivuindikaattori on toteutettu Stack Overflow -sivustolta löytyneellä yksinkertaisella koodilla, missä sivuindikaattori asetetaan seuraamaan sivuselainkomponenttia (ViewPager). Indikaattorin aktiiviselle pisteelle (sininen piste, aktiivinen sivu) ja normaalille pisteelle (harmaa piste, muut sivut) on määritelty tyylit piirtotiedostossa. (Stack Overflow 2013.)

Näkymän sivut koostuvat neljästä eri ulkoasutiedostosta, jossa jokaisella on oma logo kuvanäkymä-komponentissa ja kaksi tekstinäkymä-komponenttia. Tekstikenttien vasemmalla puolella olevat ikonit on asetettu vasempaan reunaan. Sivujen selaamiseen on lisätty Androidin sivumuuntaja-animaatio (zoom-out animation) visuaalisen ilmeen piristämiseksi. Sivumuuntaja (page transformer) kutistaa ja muuttaa sivun hieman läpinäkyväksi, kun sivuja selataan vierekkäisten sivujen välillä. Kun sivu lähestyy keskustaa, se palautuu takaisin normaalikokoonsa ja läpinäkyvyys poistuu. (Android Developers 2018c.)

3.1.7 Karttanäkymä

Valittuaan haluamansa reitin reittilistanäkymässä, käyttäjä siirtyy karttanäkymään (liite 3b). Kartalla näkyy käyttäjän valitsema reitti ja reitin opastinpisteet. Käynnistä -painike käynnistää GPS-paikannuksen, hakee käyttäjän sijainnin ja siirtää näkymän käyttäjän

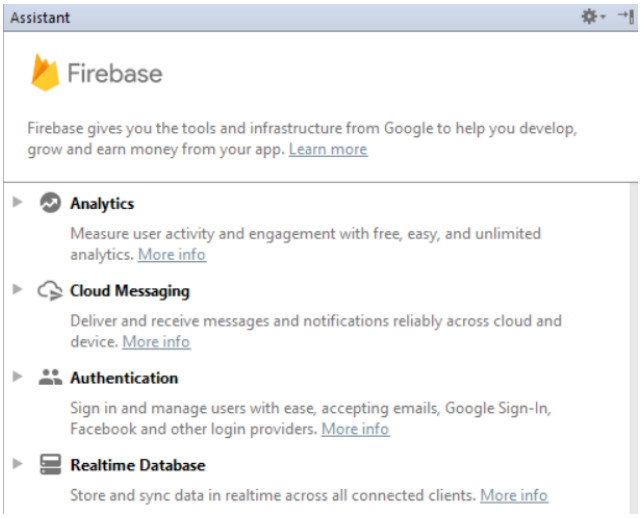
sijaintiin. Painike muuttuu Pysäytä-painikkeeksi. Yläreunan hammasratas-kuvakkeesta käyttäjä pääsee muokkaamaan muun muassa ääniopasteiden äänenvoimakkuutta, puheen korkeutta ja opasteiden etäisyyttä. Nuoli-ikoni vie takaisin reittilistanäkymään.

Karttanäkymä koostuu Mapbox-kartasta, Käynnistä- ja Pysäytä-painikkeista sekä kahdesta kuvapainikkeesta. Käynnistä- ja Pysäytä-painikkeille on määritelty taustat piirtotiedostossa, painikkeiden ikonit on asetettu vasempaan reunaan.

3.2 Taustakoodin toteutus

Sovelluksen taustakoodin kehittäminen aloitettiin yhdistämällä sovellus Firebase-tietokantaan. Tietokantaa ei ollut tarpeellista luoda erikseen tätä projektia varten, sillä toimeksiantajalla oli jo olemassa oleva Firebase-tietokanta, ja Android-sovelluksen haluttiin käyttävän samaa tietokantaa iOS-sovelluksen kanssa.

Sovelluksen yhdistäminen tietokantaan oli suoritettava ennen muiden toiminnallisuuksien kehittämistä, sillä sovelluksen kannalta tärkeät tiedot ja toiminnot, kuten sisäänkirjautuminen ja käyttäjän reittien tulostaminen, vaativat tietokannan käyttöönottoa. Android Studio -ohjelmointiympäristöön sulautetun Firebase Assistant -työkalun ansiosta (kuva 3) tietokannan käyttöönotto oli vaivatonta ja helppoa. Firebase Assistant -työkalu kuvaa lyhyesti ja ytimekkäästi tietokannan tarjoamat ominaisuudet, kuten käyttäjän varmentamisen, reaaliaikaisen tietokannan ja analyysityökalun ja sisältää pikaoppaan ominaisuuksien käyttöönottoon.



Kuva 3. Android Studion Firebase Assistant -työkalu

Tietokantaan yhdistämisen jälkeen rakennettiin GPS-käyttöoikeus -näkymän ja sisäänkirjautumisnäkymän taustakoodit. Näiden kahden näkymän toiminnallisuus oli rakennettava

ensimmäiseksi, sillä tietokannan sääntöihin on asetettu rajoite, jonka mukaan ainoastaan todennettu käyttäjä voi nähdä omat tietonsa.

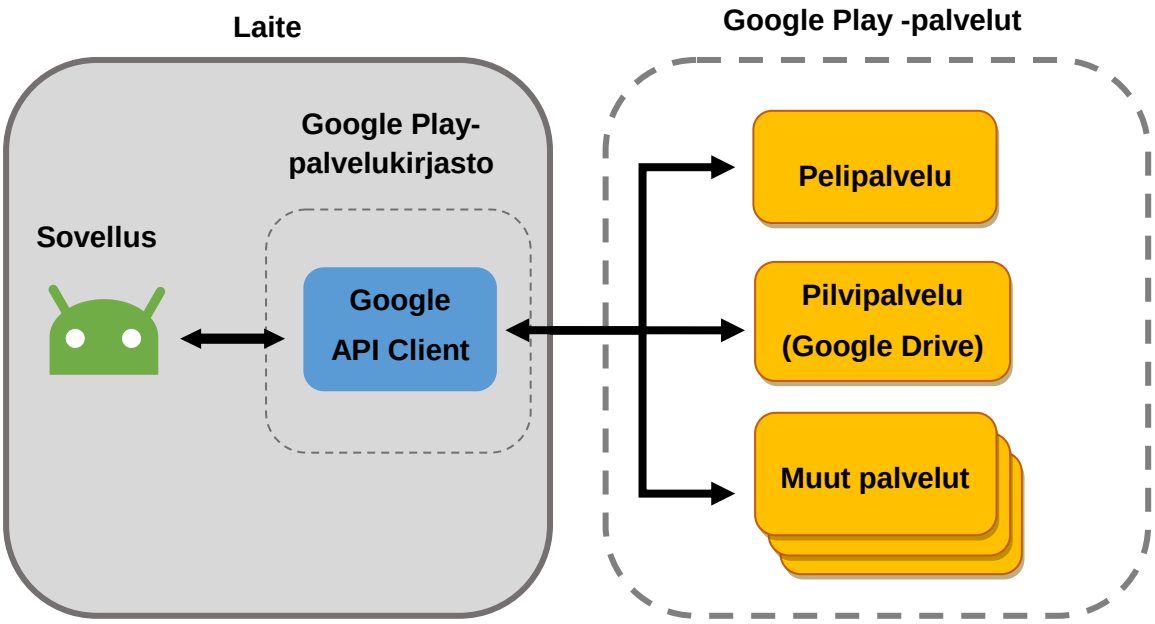
3.2.1 GPS-käyttöoikeus -näkymän toiminnallisuus

Sovelluksen luomiskutsun (onCreate) aikana alustetaan näkymän komponentit: otsikkokenttä, selitetekstikenttä sekä Hyväksy-painike. Otsikon ja tekstikentän teksteille asetetaan fontit tarvikekansion fonttitiedostosta. Lopuksi otetaan yhteys Google-sijaintipalveluun (LocationServices) Google API Client -olion avulla (kuva 4).

```
private synchronized void setUpGClient() {
    googleApiClient = new GoogleApiClient.Builder( context: this)
            .enableAutoManage( fragmentActivity: this,  i: 0,  onConnectionFailedListener: this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
    googleApiClient.connect();
}
```

Kuva 4. Google API Client -yhdistäminen ja sijaintipalvelurajapinnan lisääminen

Olion avulla on mahdollista hyödyntää Google Play -palvelukirjastoon sisältyviä ohjelmointirajapintoja, kuten Googlen sisäänkirjautumis- ja sijaintipalvelua. Google API Client mahdollistaa pääsyn Google Play -palveluiden käyttöön ja hallitsee verkkoyhteyttä käyttäjän laitteen ja Google-palvelun välillä (kuvio 5). (Google Developers 2018.)



Kuvio 5. Google API Client -olion toiminta laitteen ja palveluiden välillä (Google Developers 2018)

Google API Client -yhdistämiskutsussa (onConnect) alustetaan Hyväksy-painike ja asetetaan sille kuuntelija. Painiketta painettaessa sijainninhallinta (LocationManager) tarkistaa, onko GPS-paikannus laitteessa päällä. Jos on, sovellus ilmoittaa siitä käyttäjälle ilmoituksella ja siirtää käyttäjän valintanäkymään. Jos GPS-paikannus ei ole päällä, näytetään käyttäjälle sijaintiasetuspyyntö (displayLocationSettingsRequest). Sijaintiasetustulos -metodissa (OnActivityResult) käsitellään pyyntö käyttäjän valinnan perusteella; OK-painiketta painettaessa GPS-paikannus kytketään päälle ja käyttäjä siirretään valintanäkymään, Peruuta-painiketta painettaessa sijaintiasetuspyyntö-ikkuna suljetaan ja käyttäjälle annetaan ilmoitus GPS-paikannuksen tarpeellisuudesta.

3.2.2 Sisäänkirjautumisnäkymän toiminnallisuus

Tämän näkymän toiminnallisuus rakennettiin yhdistämällä Android Studio -ohjelmointiympäristön sisäänkirjautumis- ja tietokannan varmennus -mallikoodit toisiinsa.

Näkymän luontikutsun aikana alustetaan näkymän komponentit, kuten kuvat, sähköpostin ja salasanan tekstinsyöttökentät, Muista minut -valintaruutu, salasanan piilotuspainike (silmäikoni) sekä Kirjaudu sisään -painike. Komponenttien lisäksi alustetaan näkymän siirtymäanimaatiot, tietokannan varmentaja (FirebaseAuth), valintaruudun toiminnallisuus, salasana-tekstikentän kuuntelija (onEditorActionListener) sekä painikkeiden kuuntelijat (onClickListener) silmäikoni- ja Kirjaudu sisään -painikkeille.

Sähköpostin ja salasanan syöttämisen helpottamiseksi näkymään rakennettiin toiminnallisuus syötettyjen tietojen tallentamiseksi laitteelle jaettujen preferenssien (Shared Preferences) avulla. Muista minut -valintaruudun ollessa aktiivinen, tekstikenttiin syötetyt arvot tallennetaan ja sovellus muistaa ne, kunnes valintaruutu klikataan pois päältä. Kirjoitusvirheiden tarkistamista varten salasana-kentän oikeaan reunaan lisättiin painike, jota painamalla käyttäjä saa salasanan näkymään joko piilotettuna tai tekstimuotoisena. Salasana-tekstikenttään rakennettiin muutosten kuuntelija, joka reagoi silmäikoni-painikkeen tilaan joko piilottamalla tai näyttämällä kentän arvon. Painiketta painamalla sen tila muuttuu ei-aktiivisesta (harmaa) aktiiviseksi (sininen) ja takaisin.

Kirjaudu sisään -painikkeeseen rakennettiin kuuntelija, joka kutsuu sisäänkirjautumismetodia (attemptLogin), kun painiketta on painettu. Sisäänkirjautumismetodi tarkistaa aluksi vahvistusmetodin (validateForm) avulla, onko tekstikenttiin syötetty mitään arvoja. Jos kentät ovat tyhjiä, sovellus antaa siitä virheviestin käyttäjälle, muussa tapauksessa metodi jatkaa sisäänkirjautumisen yrittämistä (kuva 5).

```
mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener( activity: this, (task) → {
                if (task.isSuccessful()) {
                    // Sign in success, update UI with the signed-in user's information
                    FirebaseUser user = mAuth.getCurrentUser();
                    if (user != null) {
                        if (user.isEmailVerified()) {
                            updateUI(user);
                            startActivity(new Intent( packageContext: LogIn.this, Routelist.class));
                            overridePendingTransition(R.anim.fade_in, R.anim.fade_out);

                        } else {
                            // if email is not verified, send verification email to users email address.
                            user.sendEmailVerification();
                            // show alert window to inform user to go and verify their email before trying logging in
                            AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder( context: LogIn.this);
                            alertDialogBuilder.setTitle("Email not verified");
                            alertDialogBuilder.setMessage("We've sent a verification email to '" + user.getEmail() + "'. "
                                    + "Please verify your email address and try logging in again.")
                                    .setCancelable(false)
                                    .setPositiveButton( text: "OK",
                                            (dialog, id) → {
                                                    dialog.cancel();
                                            });
                            AlertDialog alert = alertDialogBuilder.create();
                            alert.show();

                            updateUI( user: null);
                        }
                    }

                } else {
                    // If sign in fails, display a message to the user.
                    Toast.makeText( context: LogIn.this, "Email or password is incorrect. Check spelling and try ...",
                            Toast.LENGTH_LONG).show();
                    updateUI( user: null);
                }
        });
```

Kuva 5. Sisäänkirjautuminen FirebaseAuth -varmentajan avulla

Tietokannan varmentajan avulla syötetyt arvot tarkistetaan ja katsotaan, löytyykö tietokannasta syötettyjä arvoja vastaava tili. Jos käyttäjän antamat tiedot vastaavat tietokannan tietoja, tarkistetaan silloin myös annetun sähköpostiosoitteen varmennuksen tila. Jos sähköpostiosoitetta ei ole varmennettu, käyttäjälle lähetetään varmennuslinkki sähköpostiin ja näytetään viesti varmennussähköpostin lähettämisestä. Käyttäjä pääsee kirjautumaan sisään, kun hän on varmentanut sähköpostiosoitteensa. Tietojen varmentamisen jälkeen sisäänkirjautumislomake piilotetaan, näytetään lataus-animaatio ja siirretään käyttäjä reittilistanäkymään. Tietojen ollessa väärin tai jos syötetyillä tiedoilla ei löydy käyttäjää, sovellus antaa siitä ilmoituksen käyttäjälle.

3.2.3 Reittilistanäkymän toiminnallisuus

Näkymän luomisvaiheen (onCreate) aikana alustetaan kirjautunut käyttäjä sekä näkymän komponentit: yläpalkki, sivuvalikko ja sen kuuntelija (DrawerListener), sivuvalikon osien kuuntelija (NavigationItemSelectedListener) sekä keskusteluikkuna (ProgressDialog). Reittien haku tietokannasta kestää pienen hetken, jolloin reittilista näyttää hetken aikaa tyhjältä. Keskusteluikkuna kertoo käyttäjälle, että reittejä haetaan tietokannasta. Ikkuna poistuu heti, kun sisältö on latautunut kokonaan.

Näkymän käynnistysvaiheessa (OnStart) alustetaan listakomponentti ja otetaan yhteys tietokantaan. Reiteistä luotu lista liitetään listasovittimeen (arrayAdapter). Tietokantakysely on yhdistetty tapahtumakuuntelijaan (childEventListener), jolloin tietokannan muutokset päivittyvät listaan välittömästi. Reittien tiedot haetaan erillisestä luokasta (class).

Jokaiselle reitille luodaan yksilöivä tunniste eli id silloin, kun reitti luodaan web-sovelluksessa ja tallennetaan tietokantaan. Tietokannasta haetusta tilannekuvasta (dataSnapshot) haetaan jokaisen tietokannassa olevan reitin tunniste hakukomennolla (getKey), ja nämä tunnisteet tallennetaan tunnistelistaan. Reitin tunnistetta tarvitaan valitun reitin tunnistamiseen; tunnisteen avulla karttanäkymään saadaan välitettyä oikean reitin tunniste, jonka avulla valitun reitin karttamerkit ja reittiviivat haetaan karttanäkymään. Reittilistaan on lisäksi laitettu osiokuuntelija (onItemClickListener), joka kuuntelee listan osioita. Kun käyttäjä valitsee haluamansa reitin listasta, valitun reitin tunniste haetaan tunnistelistasta ja välitetään karttanäkymälle aikomus-metodin avulla (Intent) samalla kun käyttäjä siirretään karttanäkymään (kuva 6).

```
// init listView and database and get current user's routes in a list
routelistView = findViewById(R.id.lv);
refdb = FirebaseDatabase.getInstance().getReference().child("users").child(user.getUid()).child("routes");
refdb.keepSynced( b: true);
arrayAdapter = new ArrayAdapter<String>( context: this, R.layout.item_row, R.id.route_name, testilista);
routelistView.setAdapter(arrayAdapter);

refdb.addChildEventListener(new ChildEventListener() {
    @Override
    public void onChildAdded(final DataSnapshot dataSnapshot, String s) {

        // get id value for each route and put values in arraylist
        routekey = dataSnapshot.getKey();
        routeKeys.add(routekey);

        final Route route = dataSnapshot.getValue(Route.class);

        // add values to list items
        if (route != null) {
            testilista.add(route.getTitle() + "\n" + route.getCreated() + "\n"+ ( String.format("%.0f", route.getLength()) ) + " m");
            arrayAdapter.notifyDataSetChanged();
            dialog.dismiss(); // close "Fetching" dialogue when all routes have been added to the list
        }

        // set click listener to list items
        routelistView.setOnItemClickListener((parent, view, position, id) -> {

                // Get chosen route's ID from routeKeys- list
                String currentRoute = routeKeys.get(position);

                // Send chosen route's ID to MapView
                Intent i = new Intent( packageContext: Routelist.this, RouteMap.class);
                i.putExtra( name: "RouteID", currentRoute);
                startActivity(i);
                overridePendingTransition(R.anim.fade_in, R.anim.fade_out);
        });
    }
```

Kuva 6. Reittilistan alustus, tietokantahaku ja reitin tunnisteen välitys karttanäkymälle

Sivuvalikon yläpalkissa näytetään käyttäjän sähköpostiosoite. Valikon osien valitsinkuuntelija (onNavigationItemsSelected) kuuntelee valikon osioita ja siirtää käyttäjän valittuun kohteeseen. Tällä hetkellä sivuvalikon osioista ovat käytössä vain ohjenäkymä (Instructions) ja uloskirjautuminen (Log out), sillä muissa osioissa ei ole vielä sisältöä. Uloskirjau-

tumismetodi (SignOut) kirjaa käyttäjän ulos ja siirtää käyttäjän takaisin sisäänkirjautumisnäkymään.

3.2.4 Karttanäkymän toiminnallisuus

Näkymän luomisvaiheessa alustetaan kirjautuneen käyttäjän tiedot (initUser -metodi), Mapbox-kartan avain (accessToken), karttanäkymä sekä reittilistalta saatu reitin tunniste (routeId). Tässä vaiheessa kutsutaan myös karttaa (onMapReady) ja suoritetaan tietokantakysely valitun reitin opastinpisteistä (Pep-Points). Opastinpisteille haetaan leveys- ja pituusasteet sekä pisteille asetetut opastetekstit ja luodaan näistä arvoista karttamerkit kartalle (kuva 7). Karttamerkkien asettamisen jälkeen kutsutaan metodia (setPolylines), jonka avulla piirretään reittiviivat karttamerkkien välille. Metodi hakee tietokannasta valitun reitin reittiviivat sekä niiden sijainnit, luo sijainneista listan ja lisää listan viivojen sijainneista Mapbox-karttaan. Lopuksi tarkistetaan GPS-paikannuksen tila; jos GPS-paikannusta ei ole kytketty päälle, sovellus pyytää käyttäjää kytkemään kyseisen toiminnon päälle.

```
if (user != null) {
    // database ref to routes "pepPoints"- children
    pepPointRef = FirebaseDatabase.getInstance().getReference().child("users").child(user.getUid()).child("routes").child(routeId).child("pepPoints");
    pepPointRef.keepSynced( b: true);
    pepPointRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {

            for (DataSnapshot pepSnapshot : dataSnapshot.getChildren()) {
                // get message from each pepPoint
                msg = pepSnapshot.child("message").getValue(String.class);
                // set values to longitude and latitude
                Object lat = pepSnapshot.child("location").child("latitude").getValue();
                Object lng = pepSnapshot.child("location").child("longitude").getValue();

                //set lat & lng objects as double values
                final double latitude = (double) lat;
                final double longitude = (double) lng;
                final LatLng latlng = new LatLng( latitude, longitude );

                // add pepPoint markers to map
                marker = mapboxMap.addMarker(new MarkerOptions()
                        .position(latlng)
                        .title(msg)
                );
                // zoom camera to route position
                mapboxMap.animateCamera(CameraUpdateFactory.newLatLngZoom(
                        new LatLng(latitude, longitude),  zoom: 14));
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
            System.out.println("The read failed: " + databaseError.getMessage());
        }
    });

    //call method for drawing the polylines
    setPolyLines();
}
```

Kuva 7. Opastinpisteiden haku ja asettaminen karttaan

Kun karttanäkymä on valmis, näkymä tarkentuu valittuun reittiin ja näyttää valitun reitin opastinpisteet ja reittiviivat. Opastinpisteitä painamalla käyttäjä näkee pisteelle asetetun reittiopasteen. Käynnistä-painiketta painamalla sovellus tarkentaa näkymän käyttäjän nykyiseen sijaintiin. Käyttäjän sijaintia seuraava metodi (onLocationChanged) nimensä mukaisesti seuraa sijainnin muutoksia ja päivittää karttanäkymää sen mukaan.

4 Pohdinta

Tässä luvussa tarkastellaan työn lopputulosta kokonaisuudessaan: käydään läpi onnistuneita ja epäonnistuneita puolia ja pohditaan, mitä olisi voinut tehdä paremmin tai toisella tavalla. Lisäksi pohditaan tehdyn työn jatkokehitysmahdollisuuksia sekä hyödyntämiskelpoisuutta.

4.1 Tulokset

Projektin lopputuloksena syntyi sovellus, joka yhtä ominaisuutta lukuun ottamatta toimii odotetulla tavalla. Käyttäjä pystyy kirjautumaan sisään omilla tunnuksillaan, näkemään luomansa reitit reittilistassa, navigoimaan sovelluksessa ja kirjautumaan halutessaan ulos ja takaisin sisään. Käyttäjä pystyy valitsemaan haluamansa reitin listasta, näkemään oman sijaintinsa lisäksi kartalla valitsemansa reitin, sen opastinpisteet ja kulkemaan reittiä pitkin. Opasteet näkyvät tekstimuotoisena opastinpisteitä painettaessa.

Työ eteni projektisuunnitelman mukaan ja aikataulun mukaisesti, muutamia poikkeuksia lukuun ottamatta. Projektisuunnitelmasta jouduttiin poikkeamaan alussa lähdekoodiin tutustumisen suhteen; lähdekoodiin tutustuttiin projektin alkuvaiheen sijasta koko projektin ajan, mikä oli hyvä päätös, sillä projektin alkuvaiheessa lähdekoodilla ei juurikaan ollut merkitystä projektin etenemisen kannalta. Projektin loppuvaiheessa asetusnäkymän toiminnallisuuden kehittäminen jouduttiin jättämään työstä pois, sillä sovelluksen tärkeämmät ominaisuudet tarvitsivat enemmän huomiota kehitysprosessin aikana. Työstä jäi projektin loppuvaiheessa pois myös alfajulkaisuvaihe, sillä opastinpisteen läheisyyden havaitsemista ja tekstistä puheeksi -ominaisuutta ei ehditty saada valmiiksi. Lopputuloksena syntyi kuitenkin kehityskelpoinen ja helposti jatkokehitettävä sovellus.

4.2 Haasteet

Haasteellisinta projektin aikana oli pysyä aikataulussa uusien opeteltavien asioiden kanssa. Eniten haasteita asettivat taustakoodin kirjoittaminen sekä vaadittavien toiminnallisuuskokonaisuuksien rakentaminen, sillä tekijällä ei ollut ennen projektin aloittamista kovinkaan vahvaa pohjaa taustakoodin kirjoittamisessa. Projektin aihe olisi pitänyt rajata alun perin pienempään alueeseen, jotta tavoitteet olisivat olleet helpommin saavutettavissa.

Vaihtoehtoisesti olisi pitänyt aloittaa sovelluksen kehittäminen sen vaikeimmasta osiosta, eli taustakoodin rakentamisesta. Se ei kuitenkaan tuntunut kovin järkevältä ratkaisulta projektin alkuvaiheessa, sillä reittien tietoihin ei päässyt käsiksi ilman sisäänkirjautumista

ja halutun reitin valitsemista. Lisäksi tekijällä oli enemmän kokemusta käyttöliittymien rakentamisesta, joten ulkoasuun keskittyminen oli tekijälle luontevinta. Vaarana oli myös sovelluksen jääminen täysin keskeneräiseksi, jos taustakoodin rakentamiseen olisi käytetty heti projektin alussa paljon aikaa.

Firebase-tietokannan rakenne oli melko monimutkainen ja sen ymmärtämiseen kului melko paljon aikaa; tarvittavat tiedot oli haettava syvältä tietokantarakenteesta, mikä aiheutti ylimääräistä selvitystyötä ja hidasti projektin etenemistä. Tekijällä oli aikaisempaa kokemusta ainoastaan MariaDB- ja MySQL-tietokannoista, joten esimerkiksi Firebase-tietokannalle asetettavat säännöt olivat tekijälle uusi asia, jota ei projektin alussa osannut ottaa huomioon.

Projektin loppuvaiheessa haasteellisin vaihe oli käyttäjän sijainnin ja opastinpisteen sijainnin välisen havainnoinnin rakentaminen, ja lopulta tekstistä puheeksi -ominaisuuden rakentaminen jäi kiinni tästä toiminnallisuudesta. Mapbox-kartan dokumentaatioista ei löytynyt esimerkkejä tämän toiminnallisuuden rakentamisen tueksi.

4.3 Sovelluksen hyödyntämiskelpoisuus ja jatkokehitysmahdollisuudet

Opasteet ovat tällä hetkellä manuaalisesti käytettävissä, eli opastinpisteitä painamalla opastetekstit näkyvät käyttäjälle tekstimuotoisena. Sovellus on käytännössä sellaisenaan käyttökelpoinen, mutta ei kuitenkaan tällä hetkellä toimi alkuperäisen sovelluksen tavoin, joten sen jatkokehittäminen on tarpeellista. Tällä hetkellä sovellusta voi hyödyntää lähinnä reitin seuraamiseen manuaalisesti.

Sovelluksen minimivaatimusten täyttämiseksi olisi seuraavaksi rakennettava logiikka opastinpisteiden läheisyyden havaitsemiseksi. Opastinpisteille tulisi määrittää jonkinlainen toimintasäde, joka aktivoisi opastintekstien lukemisen, kun käyttäjä on tarpeeksi lähellä opastinpistettä. Opastetekstien muuntaminen puheeksi onnistunee hyödyntämällä Google-yhtiön Tekstistä puheeksi -ominaisuutta.

Kun tekstien muuntaminen puheeksi on saatu toimimaan, olisi rakennettava toiminnallisuus asetusnäkymälle. Puheen nopeuden, äänenvoimakkuuden sekä puheen aktivoitumisen etäisyyden säätimille olisi rakennettava toiminnallisuudet. Lisäksi sisäänkirjautumisnäkymään olisi rakennettava automaattinen sähköpostin varmentaja, kun käyttäjä kirjautuu sovellukseen sisään ensimmäisen kerran. Tällä hetkellä sovellus lähettää käyttäjälle vahvistuslinkin sähköpostiin ja käyttäjän on käytävä klikkaamassa sähköpostin linkkiä, ennen kuin hän voi kirjautua sovellukseen sisään.

4.4 Tekijän oman oppimisen ja osaamisen arviointi

Sovelluksen kehittämiseen ja opinnäytetyön kirjoittamiseen varattiin aikaa 3,5 kuukautta, mikä on melko lyhyt aika sovelluskehityksen näkökulmasta. Projektille varattuun aikaan nähden työ onnistui hyvin, sillä sovellus on puuttuvista ominaisuuksista huolimatta käyttö- ja kehityskelpoinen kokonaisuus. Työn laatu on hyvää, sovelluksen käyttöliittymä näyttää hyvältä ja ohjelmakoodia on kommentoitu kattavasti, jotta sovelluksen jatkokehitys olisi projektin päätyttyä mahdollisimman helppoa. Projekti eteni joutuisasti ja ongelmallisista tilanteista selvittiin ahkeralla yrittämisellä ja uusien asioiden opettelulla. Tekijällä oli paljon uutta opeteltavaa, sillä ennen projektin aloittamista tekijällä oli kokemusta ainoastaan Android-ohjelmoinnista, ja siitäkin melko vähän.

Projektin aikana tekijä sai lisää kokemusta taustakoodi-ohjelmoinnista, oppi parempia ohjelmointikäytäntöjä ja sai lisää varmuutta omista taidoistaan. Tekijän taidot projektin johtamisessa sekä projektinhallinnassa saivat myös lisää harjoitusta ja tekijä oppi uusia asioita sovelluskehityksestä ja projektin suunnittelusta. Projekti oli onnistunut ja sen aikana saavutettiin paljon henkilökohtaisia tavoitteita. Sovelluksella on paljon potentiaalia kehittyä erinomaiseksi reittiopastinsovellukseksi iOS-version rinnalle.

Lähteet

Android Developers 2018a. Dashboards - Platform versions. Luettavissa: https://developer.android.com/about/dashboards/index.html. Luettu: 4.3.2018.

Android Developers 2018b. Launch-Time Performance. Luettavissa: https://developer.android.com/topic/performance/launch-time.html. Luettu: 21.3.2018

Android Developers 2018c. Slide Between Fragments with ViewPager - Zoom-out page transformer. Luettavissa: https://developer.android.com/training/animation/screen-slide.html. Luettu: 19.2.2018.

Bessarabova, E. 12.5.2017a. To Android and Back: How to Port Your iOS App to Android and Vice Versa. Blog – Mind Studios. Luettavissa: https://themindstudios.com/blog/port-ios-app-to-android-and-vice-versa/. Luettu: 1.2.2018.

Bessarabova, E. 21.2.2017b. 6 Differences Between iOS and Android App Development: Myths vs Reality. Blog – Mind Studios. Luettavissa: https://themindstudios.com/blog/6-differences-between-ios-and-android-app-development/. Luettu: 24.2.2018

Bondarenko, A. 2017. How to Convert Android App to iOS App (and Vice Versa). Stormotion Blog. Luettavissa: https://stormotion.io/blog/how-to-convert-android-app-to-ios-app-and-vice-versa/. Luettu: 3.3.2018.

Compton, M. 31.7.2015. ViewPager Without Fragments. Luettavissa: https://www.bignerdranch.com/blog/viewpager-without-fragments/. Luettu: 19.2.2018.

Google Developers 2018. Accessing Google APIs with GoogleApiClient (deprecated) | Google APIs for Android. Luettavissa: https://developers.google.com/android/guides/google-api-client. Luettu: 29.3.2018.

Nelson, R. 5.1.2018. Global App Revenue Grew 35% in 2017 to nearly $60 Billion. Sensor Tower Blog – Data That Drives App Growth. Luettavissa: https://sensortower.com/blog/app-revenue-and-downloads-2017. Luettu: 18.2.2018.

Sinhal, A. 14.1.2017. Right Way to create Splash Screen on Android. Luettavissa: https://android.jlelse.eu/right-way-to-create-splash-screen-on-android-e7f1709ba154. Luettu: 6.2.2018.

Stack Overflow 2013. Android ViewPager with bottom dots. Luettavissa: https://stackoverflow.com/questions/20586619/android-viewpager-with-bottom-dots. Luettu: 20.2.2018.

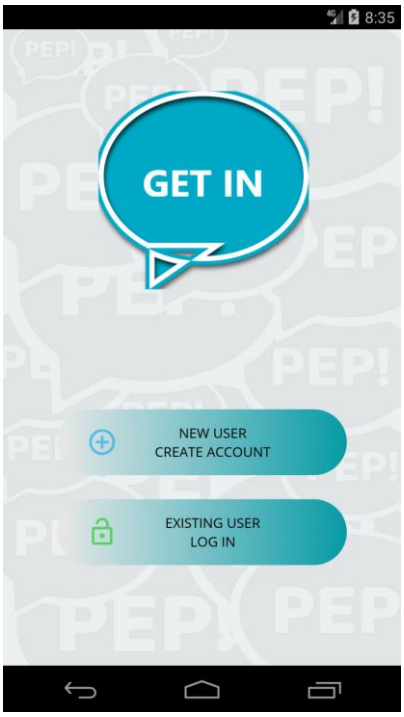
Statista 2018. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2017. Luettavissa: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/. Luettu: 8.2.2018.

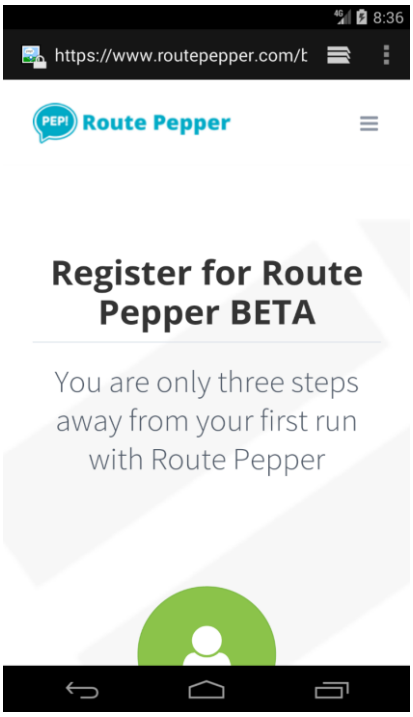
ThinkMobiles 2018. How to convert an iOS app to Android, and vice versa. Luettavissa: https://thinkmobiles.com/blog/how-convert-app/. Luettu: 1.2.2018.

Verma, A. 15.10.2017. Kotlin Programming Language Will Surpass Java On Android Next Year. Luettavissa: https://fossbytes.com/kotlin-surpass-java-android-programming/. Luettu: 5.3.2018.

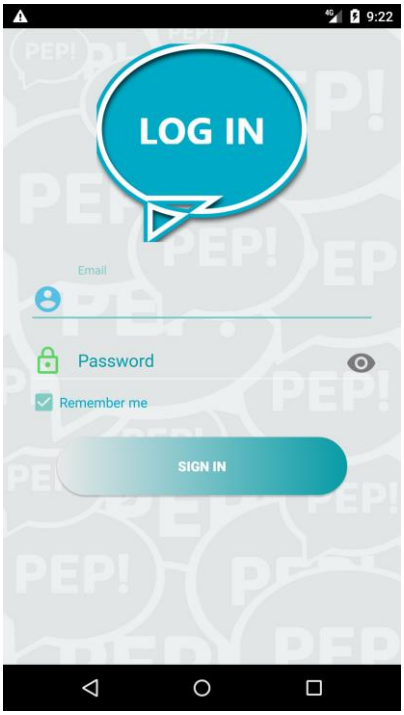
Liitteet

Liite 1a. Valintanäkymä ja Route Pepper-tilin rekisteröinti-ikkuna

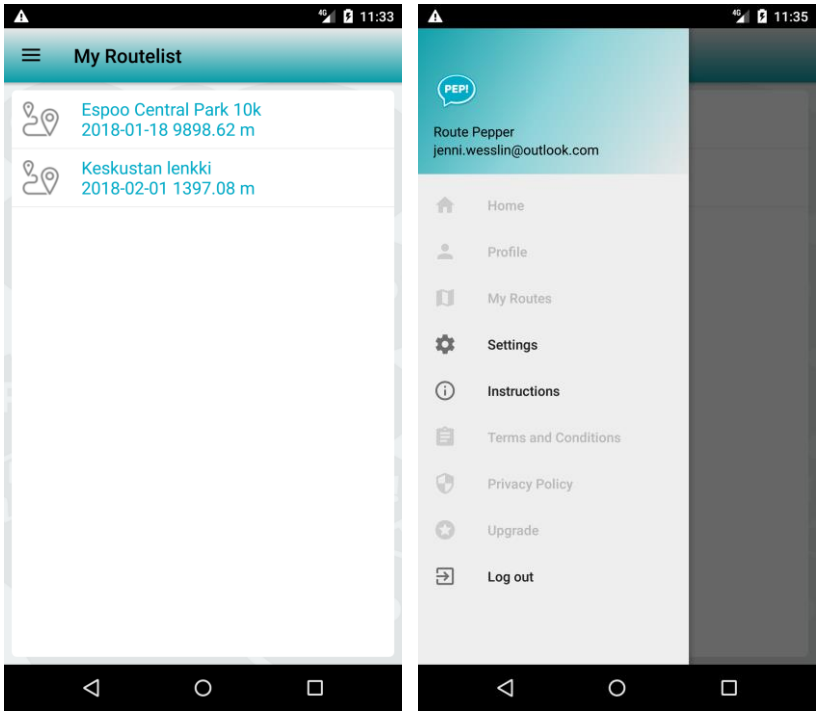




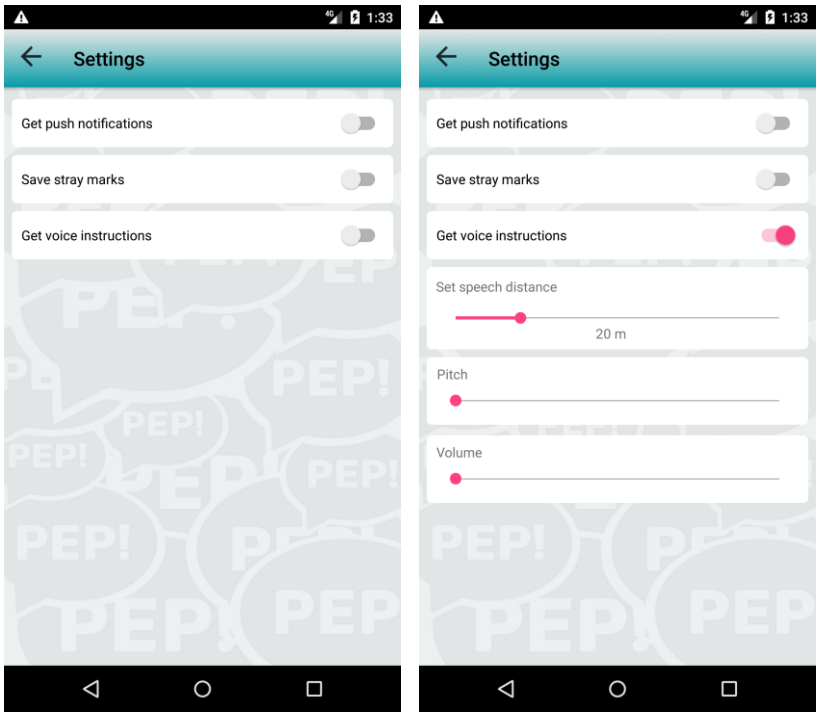
Liite 1b. Sisäänkirjautumisnäkymä



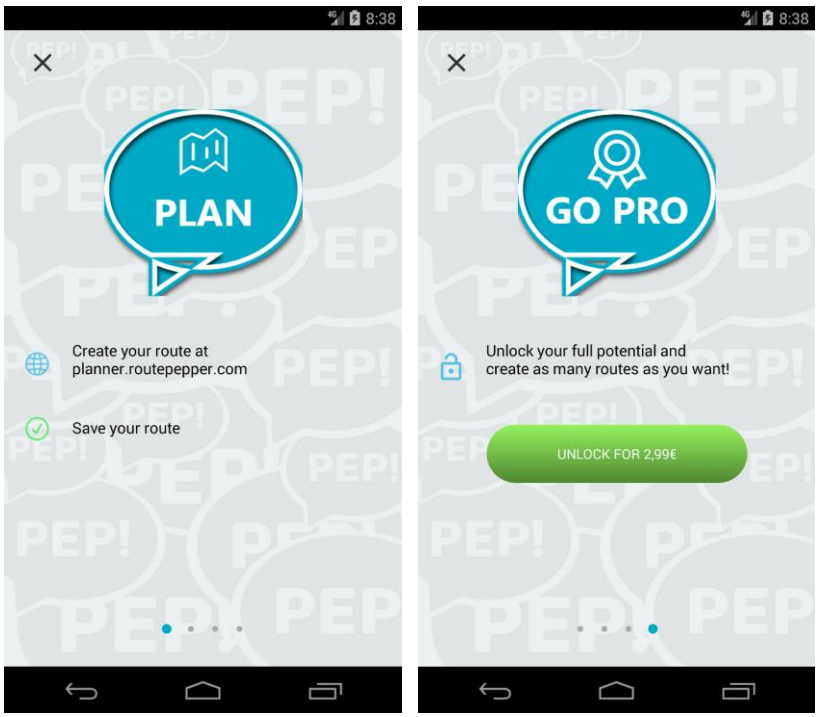
Liite 2a. Reittilistanäkymä



Liite 2b. Asetusnäkymä



Liite 3a. Ohjenäkymä



Liite 3b. Karttanäkymä